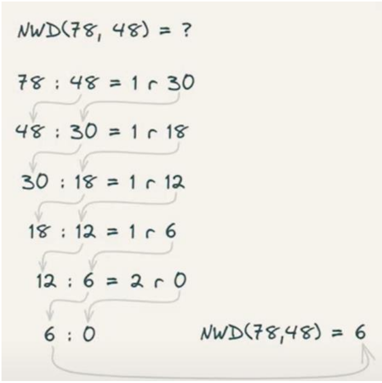
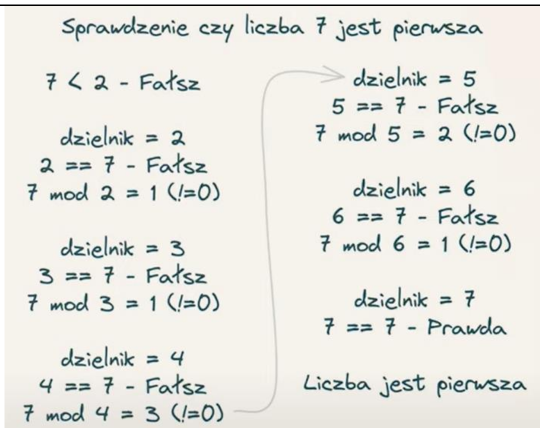
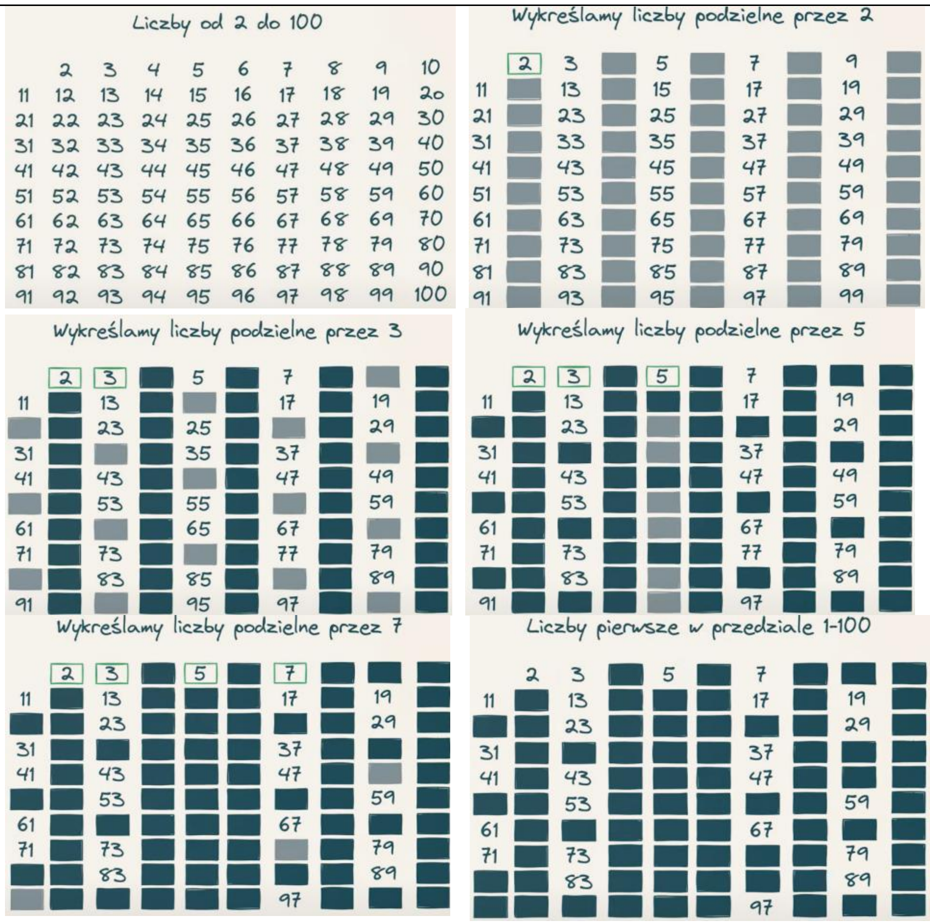
ALGORYTMY MATURALNE, KTÓRE TRZEBA ZNAĆ		
NR.	Nazwa ALGORYTMU	Co robi?
1.	**Algorytm** **Euklidesa**	1. To sposób na znalezienie **NWD największego wspólnego dzielnika dwóch liczb**. • Polega na: *wykonywaniu **kolejnych dzieleń z resztą**, gdzie za każdym razem **zmieniamy liczby miejscami**, aż jedna z nich będzie **równa 0**, a **druga liczba, to** obliczany największy wspólny dzielnik.*  2. **NWW – Najmniejsza Wspólna Wielokrotność** • NWW można obliczyć korzystając z **NWD** według wzoru, że: *NWW dwóch liczb a i b, to a * b przez NWD(a, b)* $NWW(a,b) = \frac{a*b}{NWD(a,b)}$ $NWW(78,48) = \frac{78*48}{6} = 624$
2.	**Sprawdzanie** **pierwszości liczby**	Algorytm **sprawdzania, czy liczba jest pierwsza**, polega na upewnieniu się: ***czy liczba nie dzieli się przez żadną mniejszą od siebie liczbą poza 1.*** • Najpierw sprawdzamy, **czy: liczba nie jest mniejsza od 2**, bo **0 i 1 nie są liczbami pierwszymi**. • Potem sprawdzamy: **podzielność tej liczby od dwóch do samej siebie minus 1.** • Jeśli znajdziemy liczbę, przez którą dana liczba **dzieli się bez reszty, to nie jest ona pierwsza**. Jeśli żadna liczba nie spełni tego warunku to oznacza, że liczba **jest pierwsza**

<table>
<tr><td></td><td></td><td>
</td></tr>
<tr><td>3.</td><td>Wyznaczanie n-tego wyrazu
ciągu Fibonacciego</td><td>Ciąg F. to – ciąg liczb naturalnych, gdzie każdy kolejny wyraz jest sumą dwóch poprzednich, zaczynając od tego, że pierwszymi elementami ciągu są 0 i 1.

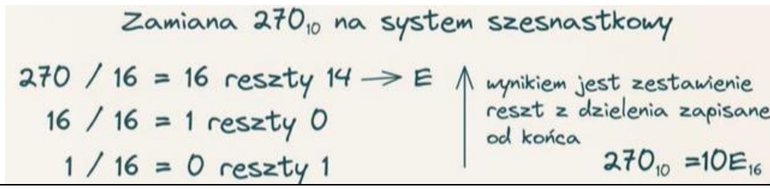
Aby obliczyć n-ty wyraz ciągu np. drugi, trzeci, czwarty itd. zaczynamy od dwóch pierwszych liczb ciągu od 0 i 1 i obliczamy kolejne wyrazy dodając do siebie dwie poprzednie liczby np. jeśli n = 2, to sumą dwóch poprzednich jest 1, a jeśli n = 3, to wynikiem jest 2. Robimy to do momentu, aż uzyskamy n-ty wyraz, który zwracamy jako wynik.
Na bazie tego algorytmu uczymy się rekurencji.

<table>
<tr><td>F0</td><td>F1</td><td>F2</td><td>F3</td><td>F4</td><td>F5</td><td>F6</td><td>F7</td><td>F8</td><td>F9</td><td>F10</td><td>F11</td><td>F12</td><td>F13</td><td>F14</td><td>F15</td><td>F16</td><td>F17</td><td>F18</td><td>F19</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>2</td><td>3</td><td>5</td><td>8</td><td>13</td><td>21</td><td>34</td><td>55</td><td>89</td><td>144</td><td>233</td><td>377</td><td>610</td><td>987</td><td>1597</td><td>2584</td><td>4181</td></tr>
</table>
F0: 0
F1: 1
F2: 0 + 1 = 1
F3: 1 + 1 = 2
F4: 1 + 2 = 3
F5: 2 + 3 = 5</td></tr>
<tr><td>4.</td><td>Sito Eratostenesa</td><td>Sito E. – to algorytm do znajdowania wszystkich liczb pierwszych w przedziale od 2 do n.

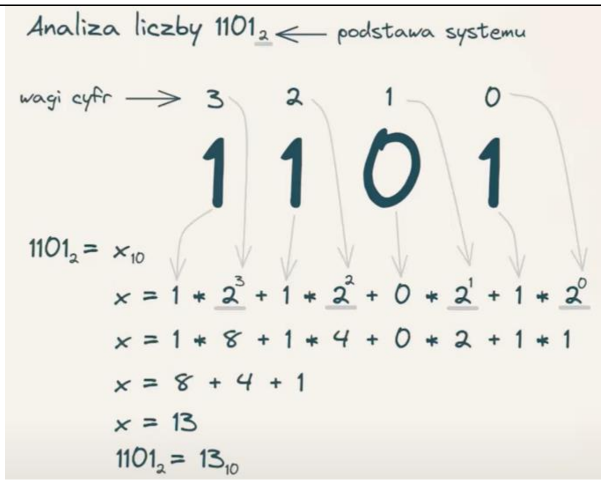
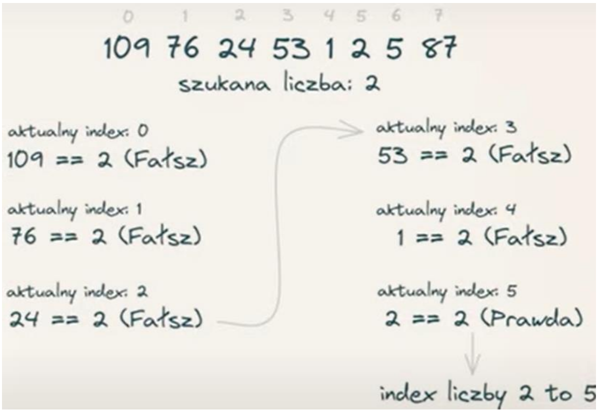
Działa on przez wykreślanie z tego zakresu liczb, które są wielokrotnościami liczb pierwszych.
Zaczynamy od najmniejszej liczby pierwszej czyli 2 i wykreślamy wszystkie jej wielokrotności.
Potem przechodzimy do następnych liczb, które nie zostały jeszcze wykreślone i znów usówamy ich wielokrotności, powtarzając ten proces aż do pierwiastka z n.
Na koniec wszystkie niewykreślone liczby są liczbami pierwszymi.
Jest on efektywniejszy, niż standardowe sprawdzanie pierwszości liczby.
</td></tr>
</table>

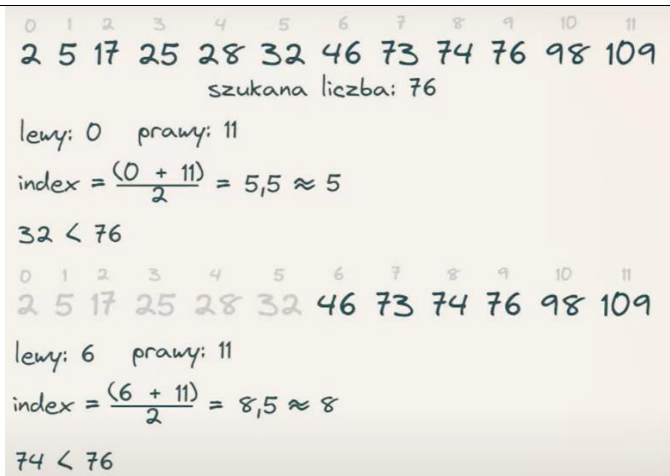
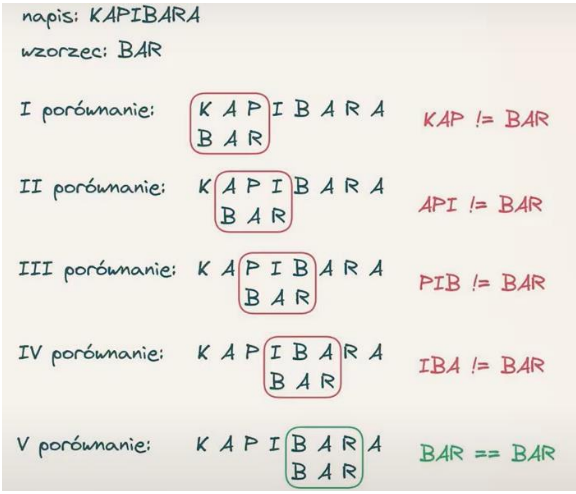
<table>
<tr>
<td></td>
<td></td>
<td>
</td>
</tr>
<tr>
<td>5.</td>
<td><u>A. Zamiana liczby</u> z systemu dziesiętnego na dowolny</td>
<td>Aby zamienić liczbę z systemu dziesiętnego na dowolny inny system dzielimy ją przez podstawę nowego systemu i zapisujemy resztę z dzielenia.

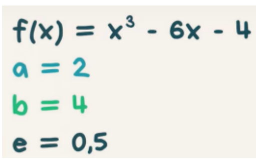
Proces powtarzamy aż liczba stanie się równa 0. Te reszty zapisane w odwrotnej kolejności tworzą liczbę w nowym systemie

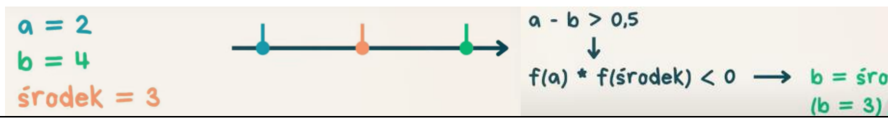

</td>
</tr>
<tr>
<td>6.</td>
<td><u>B. Zamiana liczby</u> z systemu dowolnego na dziesiętny (czyli odwrotnie jak powyżej)</td>
<td>Aby to zrobić przechodźmy po cyfrach liczby i każdą cyfrę mnożymy przez podstawę systemu podniesioną do potęgi odpowiadającej pozycji cyfry.

Pierwsza cyfra od końca ma potęgę 0, kolejna 1 itd.
Następnie sumujemy wyniki potęgowania i mnożenia, aby otrzymać zamienioną liczbę na system dziesiętny
</td>
</tr>
</table>

		
7.	**A. Przeszukiwanie liniowe**	To zagadnienie pojawia się w kontekście **złożoności obliczeniowej**. Przeszukiwanie liniowe – **najprostsze możliwe przeszukiwanie tablicy**. • *Przechodzimy po kolei po każdym elemencie tablicy i sprawdzamy czy jest on równy elementowi, którego szukamy. Można zakończyć działanie, kiedy odnajdzie się szukaną wartość.* • Przeszukiwanie liniowe ma złożoność liniową 
8.	**B. Przeszukiwanie binarne**	Jest to algorytm – **do przeszukiwania tablicy TYLKO POSORTOWANEJ**. • Opiera się on na **zasadzie dziel i zwyciężaj**. • **Za każdym razem dzielimy tablicę na pół i wybieramy tę część, w który może znajdować się wyszukiwany element**. • Proces **powtarzamy** aż: *znajdziemy element* lub *zmniejszymy przedział do zera.* • Przeszukiwanie binarne – ma złożoność logarytmiczną, czyli działa efektywniej, niż przeszukiwanie liniowe.

<table>
<tr>
<td></td>
<td></td>
<td>
</td>
</tr>
<tr>
<td>9.</td>
<td>Wyszukiwanie wzorca w tekście metodą naiwną</td>
<td>Metoda ta polega na – przesuwaniu wzorca wzdłuż tekstu i porównaniu go z odpowiadającym mu fragmentem. Jeśli wzorzec pasuje do fragmentu, oznacza to, że wzorzec został znaleziony.
Algorytm działa dopóki nie przeanalizuje całego tekstu.

</td>
</tr>
<tr>
<td>10.</td>
<td><u>ALGORYTMY ZWIĄZANE Z</u>
<u>MATEMATYKĄ:</u>

A. Wyznaczanie miejsca zerowego funkcji</td>
<td>Wyznaczanie miejsc zerowych w funkcji.
Za pomocą metody poławiania, która polega na – stopniowym zawężaniu przedziałów, w którym znajduje się miejsce zerowe.
Podobnie jak w przeszukiwaniu binarnym dzielimy na pół i sprawdzamy, w której części znajduje się miejsce zerowe.
Proces ten powtarzamy, aż róznica między końcami przedziału będzie mniejsza od zadanej dokładności.



</td>
</tr>
</table>

		$a = 2$ $b = 3$ $środek = 2{,}5$ $a - b > 0{,}5$ $\downarrow$ $f(a) * f(środek) > 0 \longrightarrow a = śr$ $(a = 2{,}5$ $a = 2{,}5$ $b = 3$ $środek = 2{,}75$ $a - b = 0{,}5 \rightarrow$ dokładność została osiągnięta, więc przybliżone miejsce zerowe znajduje s w $f(2{,}75)$
11.	**B. Obliczanie pierwiastka kwadratowego (metodą Newtona Raphsona)**	Obliczamy pierwiastek kwadratowy liczby **poprzez iteracyjne poprawianie przybliżenia.** **Zaczynamy od początkowego przybliżenia i używamy wzoru iteracyjnego do obliczania kolejnych przybliżeń**. Proces ten powtarzamy **aż róznica między kolejnymi przybliżeniami będzie mniejsza niż zadana dokładność.** • Każde nowe przybliżenie jest obliczane na podstawie poprzedniego, co pozwala na szybsze zbliżenie się do rzeczywistej wartości pierwiastka. $n = 7$ $e = 0{,}01$ $x_0 = 3{,}5$ $x_1 = (3{,}5 + 7 / 3{,}5) / 2 = 2{,}75$ $(0{,}75)$ $\lvert 2{,}75 - 3{,}5 \rvert >= 0{,}01$ $x_0 = 2{,}75$ $x_1 = (2{,}75 + 7 / 2{,}75) / 2 \approx 2{,}64773$ $(0{,}10227)$ $\lvert 2{,}64773 - 2{,}75 \rvert >= 0{,}01$ $x_0 = 2{,}64773$ $x_1 = (2{,}64773 + 7 / 2{,}64773) / 2 \approx 2{,}64575$ $(0{,}00198)$ $\lvert 2{,}64575 - 2{,}64773 \rvert < 0{,}01$ $\sqrt{7} \approx 2{,}64575$
12.	**C. Obliczanie wartości wielomianu za pomocą schematu Hornera**	Algorytm ten służy do **szybkiego obliczania wartości wielomianu**, **zmniejsza on liczbę mnożeń do minimum**, czyli **przykładowo:** *z wielomianu mającego pięć mnożeń robimy wielomian w postaci zagnieżdżonej, który ma już tylko trzy mnożenia*. • **Aby obliczyć wartość wielomianu dla danej liczby** – *zaczynamy od* ***największego współczynnika*** *i stopniowo przetwarzamy kolejne współczynniki*. Wykonujemy operacje **mnożenia i dodawania w jednym kroku**. *Mnożymy aktualny wynik poprzez wartość zmiennej i dodajemy kolejny współczynnik, kontynuujemy ten proces aż do uwzględnienia wszystkich współczynników, co pozwala szybko uzyskać wartość wielomianu*. $W(x) = 2x^3 + x^2 + 3x + 9$ $W(x) = 2 * x * x * x + x * x + 3 * x + 9$ **5 mnożeń**

		$W(x) = 2x^3 + x^2 + 3x + 9$ $= x * (2x^2 + x + 3) + 9$ $= x * (x * (2 * x + 1) + 3) + 9$ **3 mnożenia**
13.	**A. Potęgowanie** sposobem **naiwnym**	To najprostsza metoda obliczania potęgi liczby. Działa ona w ten sposób, że: • **zaczynamy od wartości 1 i wielokrotnie mnożymy ją przez podstawę potęgi tyle razy, ile wynosi wykładnik np.** *jeśli chcemy mnożyć dwa do czwartej, to po kolei mnożymy jeden przez 2, potem przez 2 i tak dalej, aż osiągniemy liczbę mnożeń równą wykładnikowi, czyli jak w poniższym przykładzie cztery razy* • Choć to najprostsza metoda, może **być mniej wydajna dla dużych wykładników** ponieważ wymaga wielu operacji mnożenia $2^4 = 1 * 2 * 2 * 2 * 2$
14.	**B. Potęgowanie szybkie**	To efektywniejszy niż poprzedni sposób potęgowania. Algorytm ten polega na: • zmniejszaniu operacji liczby mnożeń, dzięki czemu działa znacznie szybciej. • ***Dzielimy wykładnik na mniejsze części*** *i obliczamy potęgowanie rekurencyjne dla mniejszych części. Jeśli wykładnik jest* ***nieparzysty dodajemy dodatkowe mnożenia*** *poprzez podstawę do rezultatu*. • Przykładowo $4^{10} = 16^5 = 256^2 * 16 = 65536^1 * 16 = 1048576$ $4^{10} = (4 * 4) * (4 * 4) * (4 * 4) * (4 * 4) * (4 * 4)$ $= (4 * 4)^5 = 16^5$ $16^5 = (16 * 16) * (16 * 16) * 16 = (16 * 16)^2 * 16$ $= 256^2 * 16$ $256^2 * 16 = 65536^1 * 16$ $4^{10} = 65536^1 * 16 = 65536 * 16 = 1048576$
15.	**C. Potęgowanie modulo**	Algorytm ten polega na: • obliczaniu reszty z dzielenia potęgi danej liczby przez inną liczbę. **Potęgowanie naiwne modulo** $26^5 \% 33 = (((((((((1 * 26) \% 33) * 26) \% 33) * 26) \% 33) * 26) \% 33) * 26) \% 33 =$ $= (((((((26 * 26) \% 33) * 26) \% 33) * 26) \% 33) * 26) \% 33 =$ $= ((((((676 \% 33) * 26) \% 33) * 26) \% 33) * 26) \% 33 =$ $= (((((16 * 26) \% 33) * 26) \% 33) * 26) \% 33 =$ $= ((((416 \% 33) * 26) \% 33) * 26) \% 33 =$

		$= (((20 * 26)\ \%\ 33) * 26)\ \%\ 33 =$ $= ((520\ \%\ 33) * 26)\ \%\ 33 =$ $= (25 * 26)\ \%\ 33 =$ $= 650\ \%\ 33 =$ $= 23$ $26^5\ \%\ 33 = ((((26 * 26)\ \%\ 33) * ((26 * 26)\ \%\ 33))\ \%\ 33) * 26)\ \%\ 33 =$ $= ((((26 * 26)\ \%\ 33)^2\ \%\ 33) * 26)\ \%\ 33 =$ $= (((676\ \%\ 33)^2\ \%\ 33) * 26)\ \%\ 33 =$ $= ((16^2\ \%\ 33) * 26)\ \%\ 33 =$ $= ((256\ \%\ 33) * 26)\ \%\ 33 =$ $= (25 * 26)\ \%\ 33 =$ $= 650\ \%\ 33 =$ $= 23$
16.	**Wydawanie reszty z użyciem** **algorytmu zachłannego**	Zaczynamy od założeń, ż**e mamy nieskończoną liczbę monet o określonych nominałach** **chcemy wydać konkretną kwotę używając jak najmniejszej liczby** monet. • Przykładowo chcemy wydać 63 grosze, nominały którymi dysponujemy to: 1gr, 2gr, 5gr, 10gr, 20gr, 50gr • Algorytm polega na tym, że – *najpierw wybieramy monetę* ***o największym nominale****, która nie przekracza pozostałej kwoty do wydania, postępujemy tak. dopóki nie uzbieramy całej kwoty* nominały: 1 2 5 10 20 50 reszta: 63 groszy 1. wybieramy monetę 50 $(50 <= 63)$ $63 - 50 = 13$ 2. wybieramy monetę 10 $(10 <= 13)$ $13 - 10 = 3$ 3. wybieramy monetę 2 $(2 <= 3)$ $3 - 2 = 1$ 4. wybieramy monetę 1 $(1 <= 1)$ $1 - 1 = 0$ wynik: 50 10 2 1
17.	**Odwrotna Notacja Polska** **(ONP)**	Jest ona metodą – **zapisu wyrażeń arytmetycznych bez zastosowania nawiasów**. • W tej notacji **symbole operacji występują po argumentach** W poniższej tablicy widać **przykład wyrażeń w ONP**

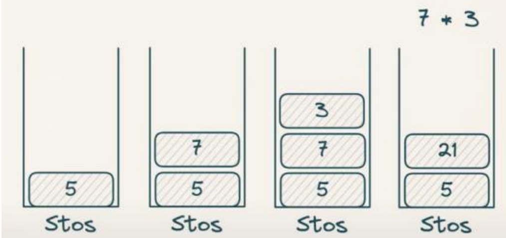
Normalna notacja	ONP
1 + 3	1 3 +
6 + 4 * 3	6 4 3 * +
7 / (3 - 5)	7 3 5 - /
2 ^ (5 + 1) + (9 - 3)	2 5 1 + ^ 9 3 - +
(8 - 5) * (4 + 6)	8 5 - 4 6 + *

Algorytm obliczania wartości ONP działa, w ten sposób, że:

- *odczytujemy po kolei każdy znak wyrażenia; jeśli jest on* **liczbą** *to* ***zapisujemy*** *go* ***na stosie****, jeśli jest* **operatorem** *to* ***pobieramy dwie ostatnie liczby ze stosu****. Wykonujemy na nich działanie i wynik umieszczamy z powrotem na stosie. Powtarzamy to działanie tak długo dopóki nie przejdziemy przez całe wyrażenie ONP.* ***Pozostała liczba na stosie będzie naszym wynikiem****.*

Działanie: 5 + 7 * 3 - 10 ^ (2+1)

ONP: 5 7 3 * + 10 2 1 + ^ -

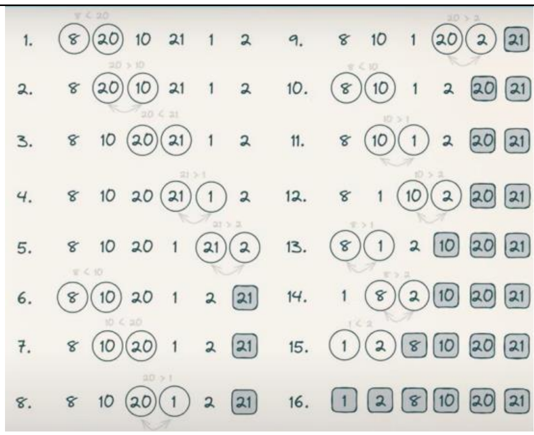
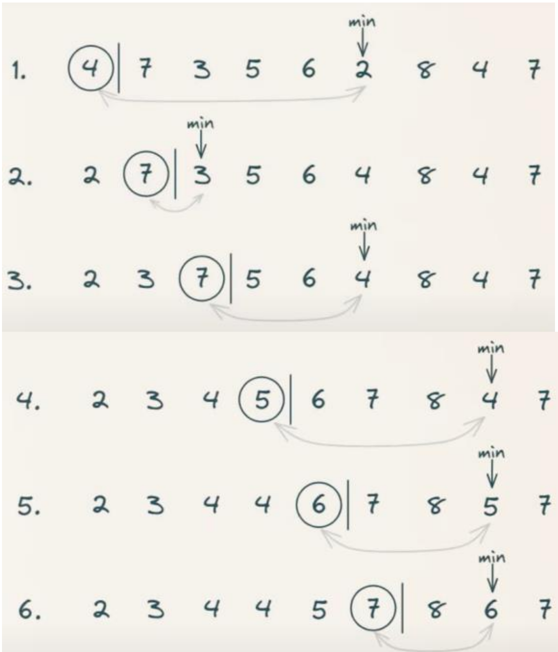


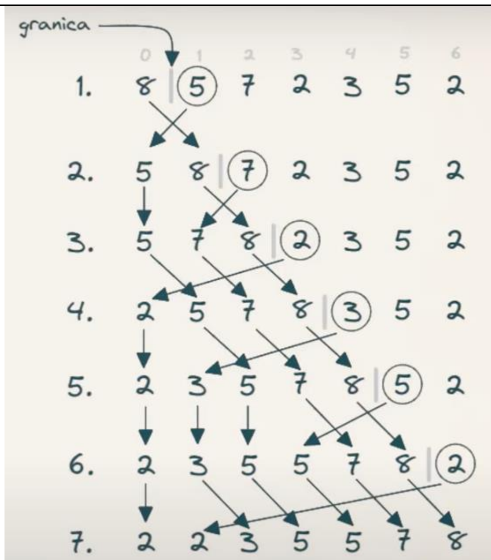
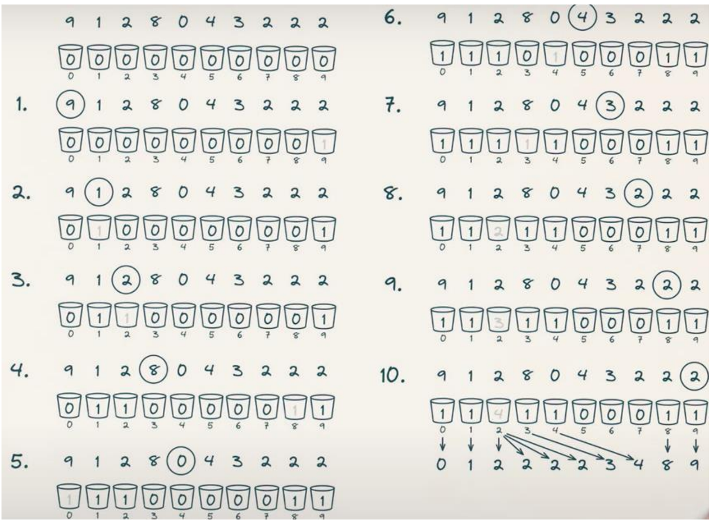
18. **A. Sortowanie bąbelkowe**

To **najprostszy** algorytm sortujący.
Polega on na:

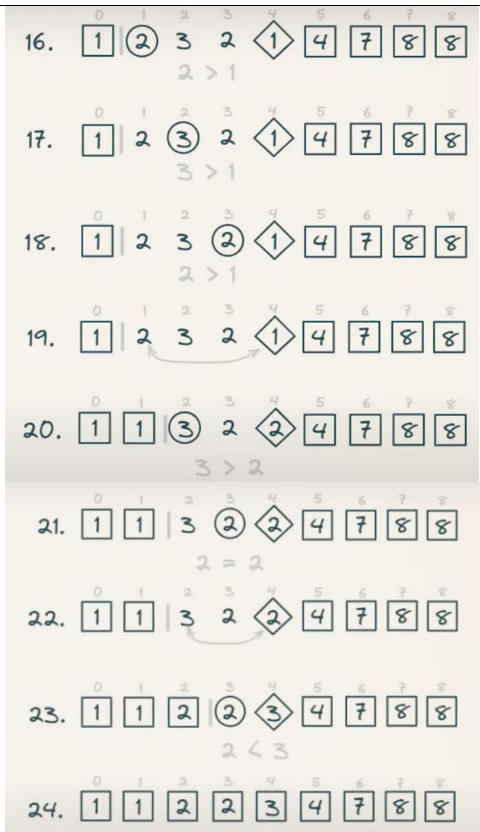
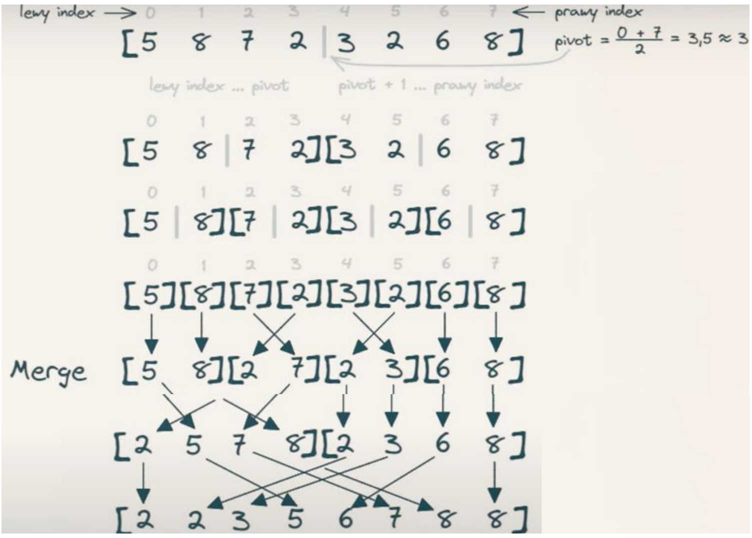
- **porównaniu ze sobą sąsiednich elementów tablicy**, dopóki nie zostanie ona posortowana.
- Dodatkowo można wykonać **operację optymalizacji**, aby zmniejszyć liczbę porównań w każdej iteracji.

Tu dodaj od siebie sposób działania algorytmu:
……
……
……
……

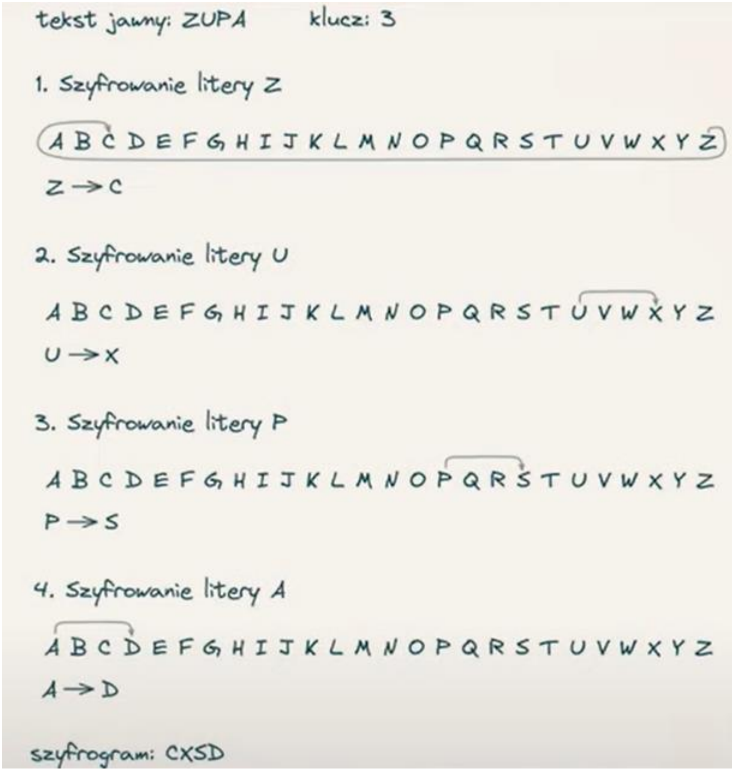
		
19.	**B.** Sortowanie **przez wybór**	Polega ono na – **znajdowaniu najmniejszego elementu po prawej stronie od aktualnego elementu**, czyli w nieposortowanej części. • *Po znalezieniu aktualny element i minimalny element **zamieniają się miejscami** tę czynność wykonujemy do momentu przejścia po całej tablicy*. 
20.	**C.** Sortowanie **przez wstawianie**	Polega ono na – **pobieraniu pierwszego elementu z prawej strony tablicy**, *czyli z części **nieposortowanej** i wstawianie go w odpowiednie miejsce w lewej części, czyli części **posortowanej.*** • Algorytm **rozpoczyna się od indeksu pierwszego,** a indeksuje aż do momentu **dojścia do końca tablicy**

		
21.	**D.** **Sortowanie** **kubełkowe**	Ma ono wiele różnych wersji. Poniższa wersja polega na – **utworzeniu liczników dla każdej wartości w tablicy i ustawieniu ich na początku na 0**. Te **liczniki to nasze kubełki**. • **Kubełki są tablicą** o długości **największej wartości w naszej tablicy plus 1**. • Przechodząc po tablicy **każde wystąpienie danej liczby** odnotowujemy zwiększając zawartość danego kubełka o **1.** Po zliczeniu wszystkich elementów wypisujemy numer danego kubełka tyle razy ile wynosi jego zawartość. W ten sposób otrzymujemy **posortowaną tablicę** 
22.	**E.** **Sortowanie** **szybkie** (Sortowania rekurencyjne)	Polega ono na wybieraniu **elementu podziału**, czyli naszego ***pivota*.** • Po **lewej stronie** od *pivota* umieszczamy **elementy od niego mniejsze** a po **prawej analogicznie większe**. Wtedy następuje **podział tablicy na mniejsze i większe elementy** od *pivota*. W tych tablicach również wybieramy *pivota* dokonuję znowu tego samego podziału i **powielamy tę czynność aż do momentu kiedy przedział jest JEDNOELEMENTOWY**.

		 lewy index prawy index granica pivot

		
23.	**F.** Sortowanie **przez scalanie** (to sortowanie rekurencyjne)	Polega ono na – **dzieleniu tablicy na dwie części, dopóki nie pozostanie jeden element.** • Powstałe **posortowane tablice łączy się w jeden posortowany ciąg**. 
24.	**A. Szyfrowanie Cezara**	To **najprostsza i jedna z najstarszych** metod szyfrowania tekstu. • **Każda litera tekstu jawnego jest zastępowana inną literą, która jest oddalona od niej o klucz, czyli o stała liczbę pozycji w alfabecie**

- Mamy **tekst jawny**, który chcemy zaszyfrować i **klucz** będący *wartością liczbową, o którą będziemy przesuwać litery*
- np. *jeśli klucz wynosi 3, to litera a stanie się literą d, litera b stanie się literą e itd.*

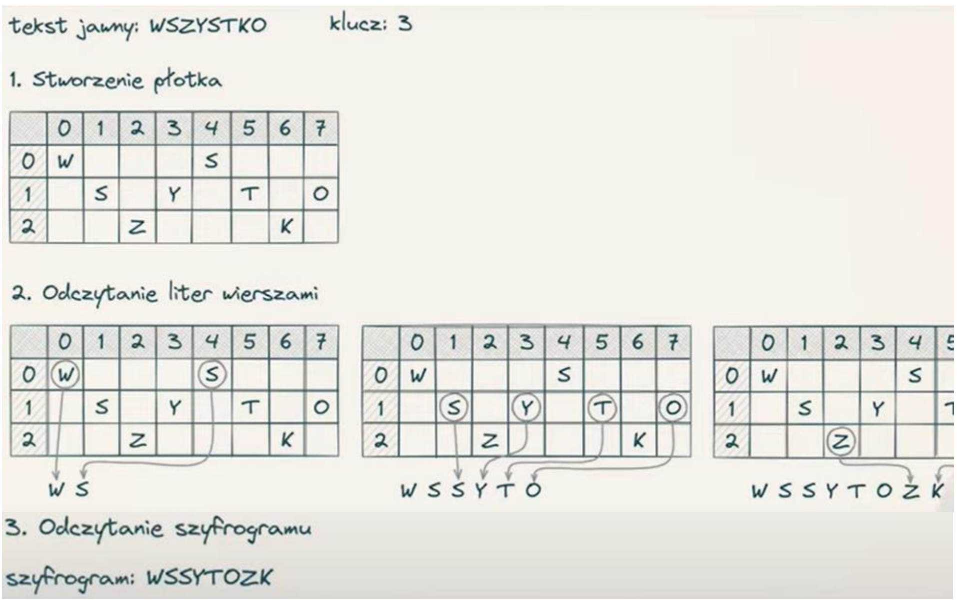


25.

B. Szyfrowanie **płotkowe**

Polega ono na – **zapisaniu tekstu jawnego w sposób przypominający płotek**

- **Wielkość płotka** jest definiowana poprzez **klucz szyfrujący**, który też jest wartością liczbową
- Następnie dzięki odpowiedniemu **odczytaniu tekstu z płotka** powstaje **szyfrogram**

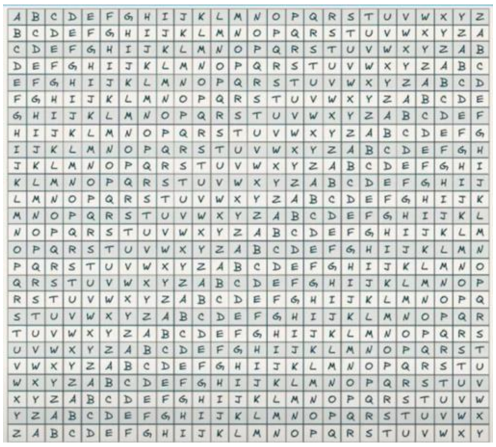


26.

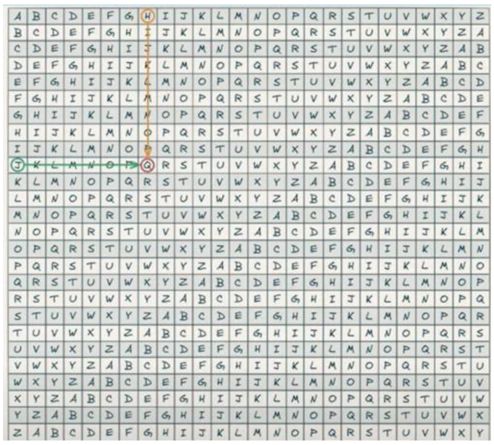
C. Szyfrowanie
Vigenere'a

Polega ono na **wykorzystaniu wielu różnych podstawień Cezarowych w zależności od litery klucza**

- Zaczynamy od **wypełnienia macierzy literami alfabetu**, następnie **każdą literę tekstu jawnego zamieniamy według klucza** przy **pomocy macierzy**, a **kluczem** w tym szyfrze jest **inne słowo**



A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y



tekst jawny: INFORMATYKA

1. Szyfrowanie litery I: $I \xrightarrow{M} U$
2. Szyfrowanie litery N: $N \xrightarrow{A} N$
3. Szyfrowanie litery F: $F \xrightarrow{T} Y$
4. Szyfrowanie litery O: $O \xrightarrow{U} I$
5. Szyfrowanie litery R: $R \xrightarrow{R} I$
6. Szyfrowanie litery M: $M \xrightarrow{A} M$
7. Szyfrowanie litery A: $A \xrightarrow{M} M$
8. Szyfrowanie litery T: $T \xrightarrow{A} T$
9. Szyfrowanie litery Y: $Y \xrightarrow{T} R$
10. Szyfrowanie litery K: $K \xrightarrow{U} E$
11. Szyfrowanie litery A: $A \xrightarrow{R} R$

szyfrogram: UNYIIMMTRER

klucz: MATURA

27.

D. Szyfrowanie
Playfair

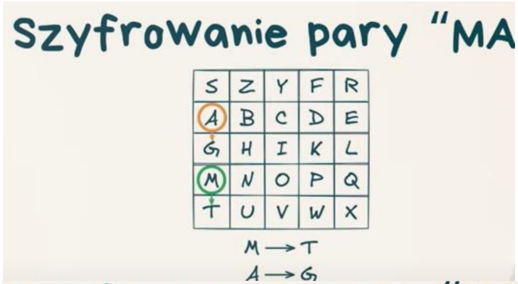
To **szyfr blokowy, który szyfruje litery parami**

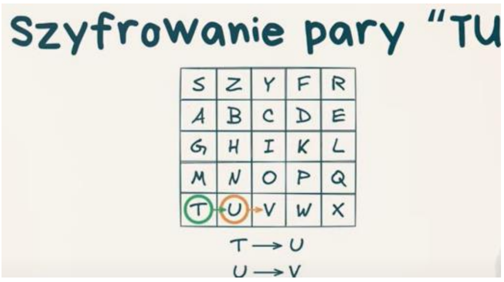
- **Klucz jest napisem** używanym do wygenerowania macierzy 5 na 5 złożonej z liter alfabetu
- **Litery są wpisane do macierzy zgodnie z kolejnością z jaką mamy w kluczu**, a **następnie w porządku alfabetycznym**
- **W macierzy** może zmieścić **się tylko 25 liter**, a alfabet angielski ma ich 26, zatem najczęściej stosuje się rozwiązanie, w którym **i** oraz j jest traktowane jako **jedna ta sama litera**
- Poniżej jest **przykładowa macierz wygenerowaną na podstawie klucza szyfr**

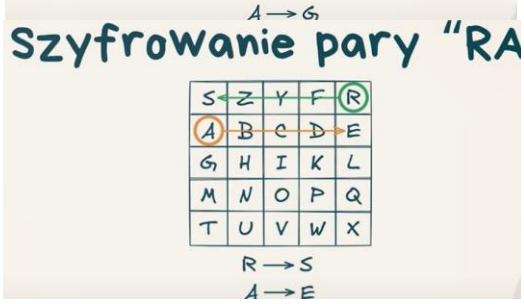
A. Jeśli litery tekstu jawnego są **w tej samej kolumnie** to *zamieniamy każdą z liter na literę znajdująca się **poniżej***

B. Jeśli litery są **w tym samym wierszu** to *zamieniamy każdą z liter na literę znajdująca się* ***po prawej stronie***

C. Jeżeli litery **tworzą prostokąt** to *zamieniamy każdą z liter na literę znajdującą się w tym samym wierszu, ale w* ***kolumnie drugiej litery z pary***







28.

<u>INNE ALGORYTMY</u>
SZYFROWANIA i SORTOWANIA

Dopisz samemu …….
1. ……….
……….
2. ……….
……….

29.

<u>Szukanie najdłuższego wspólnego podciągu</u>